```
type couleur = Rouge | Noir ;;

type bicolore = Nil | Noeud of couleur * bicolore * int * bicolore ;;
```

(* question 1 *)

```
let rec hauteurnoire = function
  | Nil                      -> 0
  | Noeud (Rouge, fg, _, _) -> hauteurnoire fg
  | Noeud (Noir, fg, _, _)  -> 1 + hauteurnoire fg ;;
```

(* question 2 *)

```
let couleur = function
  | Nil               -> Noir  (* on attribue la couleur noire Ã  l'arbre
vide *)
  | Noeud (c, _, _, _) -> c ;;

let rec rougenoir a =
  let rec aux = function
    | Nil -> (true, 0)
    | Noeud (Noir, fg, _, fd)  -> let (b1, h1) = aux fg and (b2, h2) = aux
fd in
                                  (b1 && b2 && h1 = h2, h1+1)
    | Noeud (Rouge, fg, _, fd) -> let (b1, h1) = aux fg and (b2, h2) = aux
fd in
                                  (couleur fg = Noir && couleur fd = Noir &&
b1 && b2 && h1 = h2, h1)
  in match a with
    | Noeud (Rouge, _, _, _) -> false
    | _                      -> fst (aux a) ;;
```

(* question 3 *)

```
let rec insereABR x = function
  | Nil -> Noeud (Rouge, Nil, x, Nil)
  | Noeud (c, fg, y, fd) when x < y -> Noeud (c, insereABR x fg, y, fd)
  | Noeud (c, fg, y, fd) -> Noeud (c, fg, y, insereABR x fd) ;;

(* Deux propriÃ©tÃ©s peuvent Ãªtre violÃ©es :
*)
(*   - la racine peut devenir rouge ;                                    *)
(*   - un noeud rouge peut avoir un fils rouge.                          *)
(* Dans le second cas la branche oÃ¹ a eu lieu l'insertion correspond
*)
(* Ã  un des quatre cas de figure reprÃ©sentÃ©s.
*)
```

(* question 4 *)

```
let correction_rouge = function
  | Noeud (Noir, Noeud (Rouge, Noeud (Rouge, a, x, b), y, c), z, d) ->
                                          Noeud (Rouge, Noeud (Noir, a, x,
b), y, Noeud (Noir, c, z ,d))
  | Noeud (Noir, Noeud (Rouge, a, x, Noeud (Rouge, b, y, c)), z, d) ->
                                          Noeud (Rouge, Noeud (Noir, a, x,
b), y, Noeud (Noir, c, z ,d))
```

```
  | Noeud (Noir, a, x, Noeud (Rouge, b, y, Noeud (Rouge, c, z, d))) ->
                                          Noeud (Rouge, Noeud (Noir, a, x,
b), y, Noeud (Noir, c, z ,d))
  | Noeud (Noir, a, x, Noeud (Rouge, Noeud (Rouge, b, y, c), z, d)) ->
                                          Noeud (Rouge, Noeud (Noir, a, x,
b), y, Noeud (Noir, c, z ,d))
  | a -> a ;;
```

(* question 5 *)

```
let insertion x a =
  let rec aux = function
    | Nil -> Noeud (Rouge, Nil, x, Nil)
    | Noeud (c, fg, y, fd) when x < y  -> correction_rouge (Noeud (c, aux
fg, y, fd))
    | Noeud (c, fg, y, fd)             -> correction_rouge (Noeud (c, fg,
y, aux fd))
  in match aux a with
    | Noeud (Rouge, fg, y, fd) -> Noeud (Noir, fg, y, fd)
    | b                        -> b ;;
```

(* question 6 *)

```
let rec test = function
  | 0 -> Nil
  | n -> insertion n (test (n-1)) ;;

let a = test 32768 in rougenoir a, hauteurnoire a ;;
```